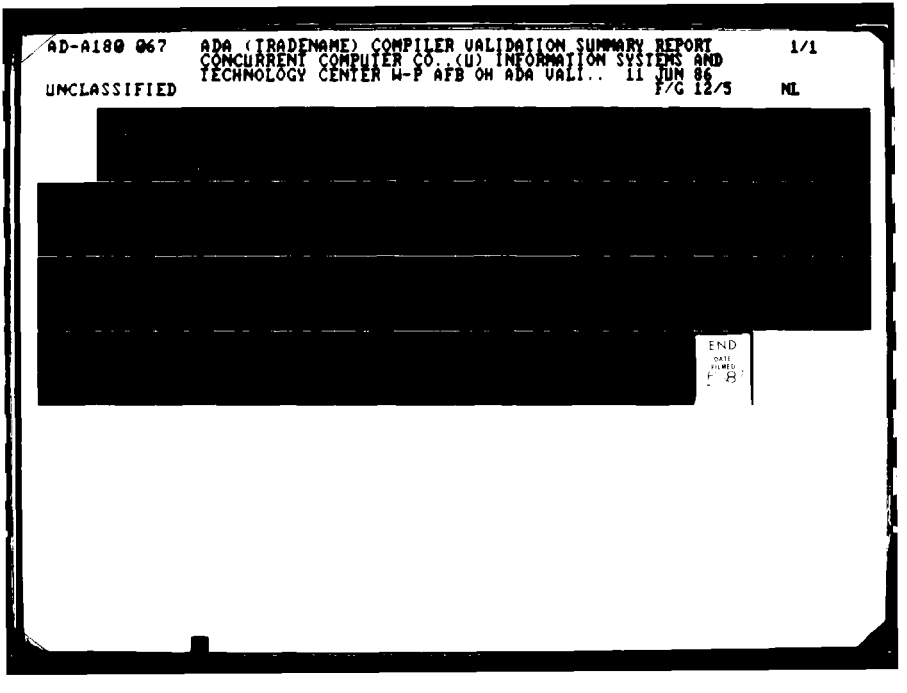
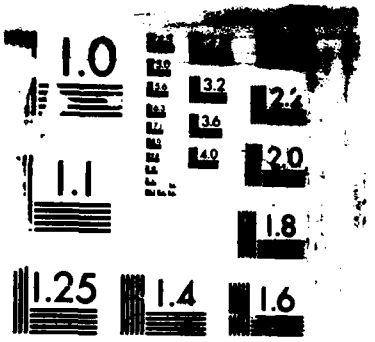

AD-A180 067
ADA (IRADENAME) COMPILER VALIDATION SUMMARY REPORT
CONCURRENT COMPUTER CO..(U) INFORMATION SYSTEMS AND
TECHNOLOGY CENTER W-P AFB OH ADA VALI.. 11 JUN 86
1/1
UNCLASSIFIED
F/G 12/5
NL
END
DATE FILMED
8


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
3.2
3.6
4.0

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETEING FORM
1. REPORT NUMBER | 2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* Ada Compiler Validation Summary Report: Concurrent Computer Corporation, C3 Ada, Version R00-00.00, Concurrent Computer Corporation Series 3200	5. TYPE OF REPORT & PERIOD COVERED 11 JUN 1986 to 11 JUN 1987 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Wright-Patterson	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION AND ADDRESS Ada Validation Facility ASD/SIOL Wright-Patterson AFB OH 45433-6503	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Ada Joint Program Office United States Department of Defense Washington, DC 20301-3081	12. REPORT DATE 11 JUN 1986 13. NUMBER OF PAGES 48
14. MONITORING AGENCY NAME & ADDRESS*(If different from Controlling Office)* Wright-Patterson	15. SECURITY CLASS *(of this report)* UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20. If different from Report)*

UNCLASSIFIED

18. SUPPLEMENTARY NOTES

19. KEYWORDS *(Continue on reverse side if necessary and identify by block number)*

Ada Programming language, Ada Compiler Validation Summary Report, Ada Compiler Validation Capability, ACVC, Validation Testing, Ada Validation Office, AVO, Ada Validation Facility, AVF, ANSI/MIL-STD-1815A, Ada Joint Program Office, AJPO

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

See Attached.

AD-A180 067

DD FORM 1473 1 JAN 73 EDITION OF 1 NOV 65 IS OBSOLETE S/N 0102-LF-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

Ada® Compiler Validation Summary Report:

Compiler Name: C³ Ada, Version R00-00.00

Host Computer:	Target Computer:
Concurrent Computer Corporation	Concurrent Computer Corporation
Series 3200	Series 3200
under	under
OS/32, Version R08-01	OS/32, Version R08-01

Testing Completed 11 JUN 1986 Using ACVC 1.7

This report has been reviewed and is approved.

Georgeanne Chitwood

Ada Validation Facility
Georgeanne Chitwood
ASD/SIOL
Wright-Patterson AFB OH

for JFK

Ada Validation Office
Dr. John F. Kramer
Institute for Defense Analyses
Alexandria VA

Virginia L. Castor

Ada Joint Program Office
Virginia L. Castor
Director
Department of Defense
Washington DC

Accession For	
NTIS GRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

®Ada is a registered trademark of the United States Government (Ada Joint Program Office).

AVF Control Number: AVF-VSR-38.0886

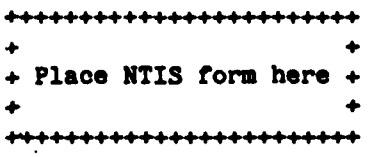
Ada® COMPILER
VALIDATION SUMMARY REPORT:
Concurrent Computer Corporation
C^3 Ada, Version R00-00.00
Concurrent Computer Corporation Series 3200

Completion of On-Site Validation:
11 JUN 1986

Prepared By:
Ada Validation Facility
ASD/SIOL
Wright-Patterson AFB OH 45433-6503

Prepared For:
Ada Joint Program Office
United States Department of Defense
Washington, D.C.

®Ada is a registered trademark of the United States Government (Ada Joint Program Office).


Place NTIS form here

EXECUTIVE SUMMARY

This Validation Summary Report (VSR) summarizes the results and conclusions of validation testing performed on the C^3_μ Ada compiler, Version R00-00.00, using Version 1.7 of the Ada® Compiler Validation Capability (ACVC).

The validation process includes submitting a suite of standardized tests (the ACVC) as inputs to an Ada compiler and evaluating the results. The purpose is to ensure conformance of the compiler to ANSI/MIL-STD-1815A Ada by testing that it properly implements legal language constructs and that it identifies and rejects illegal language constructs. The testing also identifies behavior that is implementation dependent but permitted by ANSI/MIL-STD-1815A. Six classes of tests are used. These tests are designed to perform checks at compile time, at link time, or during execution.

On-site testing was performed 4 JUN 1986 through 11 JUN 1986 at Concurrent Computer Corporation, Tinton Falls NJ, under the direction of the Ada Validation Facility (AVF), according to Ada Validation Organization (AVO) policies and procedures. The C^3 Ada compiler, Version R00-00.00, is hosted on the Concurrent Computer Corporation Series 3200 MPS operating under OS/32, Version R08-01 running the Multi-Terminal Monitor (MTM).

The results of validation are summarized in the following table:

RESULT	TEST CLASS						TOTAL
	A	B	C	D	E	L	
Passed	66	816	1121	17	9	21	2050
Failed	0	0	0	0	0	0	0
Inapplicable	2	8	199	0	2	2	213
Withdrawn	0	4	12	0	0	0	16
TOTAL	68	828	1332	17	11	23	2279

®Ada is a registered trademark of the United States Government (Ada Joint Program Office).

There were 16 withdrawn tests in ACVC Version 1.7 at the time of this validation attempt. A list of these tests appears in Appendix D.

Some tests demonstrate that some language features are or are not supported by an implementation. For this implementation, the tests determined the following:

- LONG_INTEGER and SHORT_FLOAT are not supported.
- The additional predefined types TINY_INTEGER, SHORT_INTEGER, and LONG_FLOAT are supported.
- Representation specifications for noncontiguous enumeration representations are supported.
- Generic unit specifications and bodies can be compiled in separate compilations.
- Pragma INLINE is not supported for procedures nor is it supported for functions.
- The package SYSTEM is used by package TEXT_IO.
- Modes IN_FILE and OUT_FILE are supported for sequential I/O.
- Instantiation of the package SEQUENTIAL_IO with unconstrained array types is not supported.
- Instantiation of the package SEQUENTIAL_IO with unconstrained record types with discriminants is not supported.
- RESET and DELETE are supported for sequential and direct I/O.
- Modes IN_FILE, INOUT_FILE, and OUT_FILE are supported for direct I/O.
- Instantiation of package DIRECT_IO with unconstrained array types and unconstrained types with discriminants is not supported.
- Dynamic creation and deletion of files are supported.
- More than one internal file can be associated with the same external file.
- An external file associated with more than one internal file can be reset.
- Illegal file names can exist.

ACVC Version 1.7 was taken on-site via magnetic tapes to Concurrent Computer Corporation, Tinton Falls NJ. All tests, except the withdrawn tests and any executable tests that make use of a floating-point precision greater than SYSTEM.MAX_DIGITS, were compiled on two identical Concurrent Computer Corporation 3200 MPS computers. Class A, C, D, and E tests were executed on the 3200 MPS computers. A subset of the tests was compiled and executed on the Concurrent Computer Corporation 3203, 3205, 3210, 3230, 3250, 3230 XP, 3250 XP, 3230 MPS, and 3260 MPS computers. These computers have architectures identical to the 3200 MPS and operate under the OS/32 operating system.

On completion of testing, execution results for Class A, C, D, and E tests were examined. Compilation results for Class B tests were analyzed for correct diagnosis of syntax and semantic errors. Compilation and link results of Class L tests were analyzed for correct detection of errors. The test results produced from the subset testing were identical to the results generated by the compiler operating on the 3200 MPS.

The compiler was also tested using a 3200 MPS host computer and a 3280 target computer. A subset of the tests was compiled and linked on the 3200 MPS, and the executable images were moved to the 3280 via magnetic tape and executed.

The AVF identified 2093 of the 2279 tests in Version 1.7 of the ACVC as potentially applicable to the validation of the C^3 Ada compiler, Version R00-00.00. Excluded were 170 tests requiring a floating-point precision greater than that supported by the implementation and the 16 withdrawn tests. After the 2093 tests were processed, 43 tests were determined to be inapplicable. The remaining 2050 tests were passed by the compiler.

The AVF concludes that these results demonstrate acceptable conformance to ANSI/MIL-STD-1815A.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

This Validation Summary Report (VSR) describes the extent to which a specific Ada compiler conforms to ANSI/MIL-STD-1815A. This report explains all technical terms used within it and thoroughly reports the results of testing this compiler using the Ada Compiler Validation Capability (ACVC). An Ada compiler must be implemented according to the Ada Standard (ANSI/MIL-STD-1815A). Any implementation-dependent features must conform to the requirements of the Ada Standard. The entire Ada Standard must be implemented, and nothing can be implemented that is not in the Standard.

Even though all validated Ada compilers conform to ANSI/MIL-STD-1815A, it must be understood that some differences do exist between implementations. The Ada Standard permits some implementation dependencies--for example, the maximum length of identifiers or the maximum values of integer types. Other differences between compilers result from limitations imposed on a compiler by the operating system and by the hardware. All of the dependencies demonstrated during the process of testing this compiler are given in this report.

VSRs are written according to a standardized format. The reports for several different compilers may, therefore, be easily compared. The information in this report is derived from the test results produced during validation testing. Additional testing information as well as details which are unique for this compiler are given in section 3.7. The format of a validation report limits variance between reports, enhances readability of the report, and minimizes the delay between the completion of validation testing and the publication of the report.

1.1 PURPOSE OF THIS VALIDATION SUMMARY REPORT

This VSR documents the results of the validation testing performed on an Ada compiler. Testing was carried out for the following purposes:

- To attempt to identify any language constructs supported by the compiler that do not conform to the Ada Standard

- To attempt to identify any unsupported language constructs required by the Ada Standard
- To determine that the implementation-dependent behavior is allowed by the Ada Standard

Testing of this compiler was conducted by SofTech, Inc., under the direction of the AVF according to policies and procedures established by the Ada Validation Organization (AVO). Testing was conducted from 4 JUN 1986 through 11 JUN 1986 at Concurrent Computer Corporation, Tinton Falls NJ.

1.2 USE OF THIS VALIDATION SUMMARY REPORT

Consistent with the national laws of the originating country, the AVO may make full and free public disclosure of this report. In the United States, this is provided in accordance with the "Freedom of Information Act" (5 U.S.C. #552). The results of this validation apply only to the computers, operating systems, and compiler versions identified in this report.

The organizations represented on the signature page of this report do not represent or warrant that all statements set forth in this report are accurate and complete, or that the subject compiler has no nonconformances to ANSI/MIL-STD-1815A other than those presented. Copies of this report are available to the public from:

Ada Information Clearinghouse
Ada Joint Program Office
OUSDRE
The Pentagon, Rm 3D-139
1211 S. Fern, C-107
Washington DC 20301-3081

or from:

Ada Validation Facility
ASD/SIOL
Wright-Patterson AFB OH 45433-6503

Questions regarding this report or the validation test results should be directed to the AVF listed above or to:

Ada Validation Organization
Institute for Defense Analyses
1801 North Beauregard
Alexandria VA 22311

1.3 RELATED DOCUMENTS

1. Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, FEB 1983.
2. Ada Validation Organization: Policies and Procedures, MITRE Corporation, JUN 1982, PB 83-110601.
3. Ada Compiler Validation Capability Implementers' Guide, SofTech, Inc., DEC 1984.

1.4 DEFINITION OF TERMS

ACVC	The Ada Compiler Validation Capability. A set of programs that evaluates the conformance of a compiler to the Ada language specification, ANSI/MIL-STD-1815A.
Ada Standard	ANSI/MIL-STD-1815A, February 1983.
Applicant	The agency requesting validation.
AVF	The Ada Validation Facility. In the context of this report, the AVF is responsible for conducting compiler validations according to established policies and procedures.
AVO	The Ada Validation Organization. In the context of this report, the AVO is responsible for setting policies and procedures for compiler validations.
Compiler	A processor for the Ada language. In the context of this report, a compiler is any language processor, including cross-compilers, translators, and interpreters.
Failed test	A test for which the compiler generates a result that demonstrates nonconformance to the Ada Standard.
Host	The computer on which the compiler resides.

Inapplicable test	A test that uses features of the language that a compiler is not required to support or may legitimately support in a way other than the one expected by the test.
LMC	The Language Maintenance Committee whose function is to resolve issues concerning the Ada language.
Passed test	A test for which a compiler generates the expected result.
Target	The computer for which a compiler generates code.
Test	A program that evaluates the conformance of a compiler to a language specification. In the context of this report, the term is used to designate a single ACVC test. The text of a program may be the text of one or more compilations.
Withdrawn test	A test found to be inaccurate in checking conformance to the Ada language specification. A withdrawn test has an invalid test objective, fails to meet its test objective, or contains illegal or erroneous use of the language.

1.5 ACVC TEST CLASSES

Conformance to ANSI/MIL-STD-1815A is measured using the ACVC. The ACVC contains both legal and illegal Ada programs structured into six test classes: A, B, C, D, E, and L. The first letter of a test name identifies the class to which it belongs. Special program units are used to report the results of the Class A, C, D, and E tests during execution. Class B tests are expected to produce compilation errors, and Class L tests are expected to produce link errors.

Class A tests check that legal Ada programs can be successfully compiled and executed. (However, no checks are performed during execution to see if the test objective has been met.) For example, a Class A test checks that reserved words of another language (other than those already reserved in the Ada language) are not treated as reserved words by an Ada compiler. A Class A test is passed if no errors are detected at compile time and the program executes to produce a message indicating that it has passed.

Class B tests check that a compiler detects illegal language usage. Class B tests are not executable. Each test in this class is compiled and the resulting compilation listing is examined to verify that every syntactical or semantic error in the test is detected. A Class B test is passed if every illegal construct that it contains is detected by the compiler.

Class C tests check that legal Ada programs can be correctly compiled and executed. Each Class C test is self-checking and produces a PASSED, FAILED, or NOT-APPLICABLE message indicating the result when it is executed.

Class D tests check the compilation and execution capacities of a compiler. Since there are no requirements placed on a compiler by the Ada Standard for some parameters (e.g., the number of identifiers permitted in a compilation, the number of units in a library, and the number of nested loops in a subprogram body), a compiler may refuse to compile a Class D test and still be a conforming compiler. Therefore, if a Class D test fails to compile because the capacity of the compiler is exceeded, the test is classified as inapplicable. If a Class D test compiles successfully, it is self-checking and produces a PASSED or FAILED message during execution.

Each Class E test is self-checking and produces a NOT-APPLICABLE, PASSED, or FAILED message when it is compiled and executed. However, the Ada Standard permits an implementation to reject programs containing some features addressed by Class E tests during compilation. Therefore, a Class E test is passed by a compiler if it is compiled successfully and executes to produce a PASSED message, or if it is rejected by the compiler for an allowable reason.

Class L tests check that incomplete or illegal Ada programs involving multiple, separately compiled units are detected and not allowed to execute. Class L tests are compiled separately and execution is attempted. A Class L test passes if it is rejected at link time--that is, an attempt to execute the main program must generate an error message before any declarations in the main program or any units referenced by the main program are elaborated.

Two library units, the package REPORT and the procedure CHECK_FILE, support the self-checking features of the executable tests. The package REPORT provides the mechanism by which executable tests report results. It also provides a set of identity functions used to defeat some compiler optimization strategies and force computations to be made by the target computer instead of by the compiler on the host computer. The procedure CHECK_FILE is used to check the contents of text files written by some of the Class C tests for chapter 14 of the Ada Standard.

The operation of these units is checked by a set of executable tests. These tests produce messages that are examined to verify that the units are operating correctly. If these units are not operating correctly, then the validation is not attempted.

Some of the conventions followed in the ACVC are intended to ensure that the tests are reasonably portable without modification. For example, the tests make use of only the basic set of 55 characters, contain lines with a maximum length of 72 characters, use small numeric values, and place features that may not be supported by all implementations in separate tests. However, some tests contain values that require the test to be customized according to implementation-specific values. The values used for this validation are listed in Appendix C.

A compiler must correctly process each of the tests in the suite and demonstrate conformance to the Ada Standard by either meeting the pass criteria given for the test or by showing that the test is inapplicable to the implementation. Any test that was determined to contain an illegal

language construct or an erroneous language construct is withdrawn from the ACVC and, therefore, is not used in testing a compiler. The nonconformant tests are given in Appendix D.

CHAPTER 2

CONFIGURATION INFORMATION

2.1 CONFIGURATION TESTED

The candidate compilation system for this validation was tested under the following configuration:

Compiler: C^3 Ada, Version R00-00.00

Test Suite: Ada Compiler Validation Capability, Version 1.7

Host Computer:

Machine:	Concurrent Computer Corporation 3200 MPS
Operating System:	OS/32, Version R08-01
Memory Size:	16 megabytes

Target Computer:

Machine:	Concurrent Computer Corporation 3200 MPS
Operating System:	OS/32, Version R08-01
Memory Size:	16 megabytes

In addition, the following configurations in the Concurrent Computer Corporation 3200 series operating under OS/32, Version R08-01, were tested using a subset of the ACVC tests run on the 3200 MPS:

CONFIGURATION INFORMATION

Host		Target	
Machine	Memory Size (megabytes)	Machine	Memory Size (megabytes)
3203	4	3203	4
3205	4	3205	4
3210	8	3210	8
3230	16	3230	16
3250	16	3250	16
3230 XP	8	3230 XP	8
3250 XP	16	3250 XP	16
3230 MPS	8	3230 MPS	8
3260 MPS	8	3260 MPS	8
3200 MPS	16	3280	8

2.2 CERTIFICATE INFORMATION

Base Configuration:

Compiler: C^3 Ada, Version R00-00.00

Test Suite: Ada Compiler Validation Capability, Version 1.7

Certificate Date: 16 JUL 1986

Host Computer:

Machine(s): Concurrent Computer Corporation Series 3200: 3200 MPS, 3203, 3205, 3210, 3230, 3250, 3230 XP, 3250 XP, 3230 MPS, and 3260 MPS

Operating System: OS/32, Version R08-01

Target Computer:

Machine(s): Concurrent Computer Corporation Series 3200: 3200 MPS, 3203, 3205, 3210, 3230, 3250, 3230 XP, 3250 XP, 3230 MPS, 3260 MPS, and 3280

Operating System: OS/32, Version R08-01

2.3 IMPLEMENTATION CHARACTERISTICS

One of the purposes of validating compilers is to determine the behavior of a compiler in those areas of the Ada Standard that permit implementations to differ. Class D and E tests specifically check for such implementation differences. However, tests in other classes also characterize an implementation. This compiler is characterized by the following interpretations of the Ada Standard:

- Nongraphic characters.

  Nongraphic characters are defined in the ASCII character set but are not permitted in Ada programs, even within character strings. The compiler correctly recognizes these characters as illegal in Ada compilations. The characters are not printed in the output listing. (See test B26005A.)

- Capacities.

  The compiler correctly processes compilations containing loop statements nested to 65 levels, block statements nested to 65 levels, and recursive procedures nested to 17 levels. It correctly processes a compilation containing 723 variables in the same declarative part. (See tests D55A03A through D55A03H, D56001B, D64005E through D64005G, and D29002K.)

- Universal integer calculations.

  An implementation is allowed to reject universal integer calculations having values that exceed SYSTEM.MAX_INT. This implementation does not reject such calculations and processes them correctly. (See tests D4A002A, D4A002B, D4A004A, and D4A004B.)

- Predefined types.

  This implementation supports the additional predefined types SHORT_INTEGER, LONG_FLOAT, and TINY_INTEGER in the package STANDARD. (See tests B86001CR, B86001CQ, and B86001DT.)

- Based literals.

  An implementation is allowed to reject a based literal with a value exceeding SYSTEM.MAX_INT during compilation, or it may raise NUMERIC_ERROR during execution. This implementation raises NUMERIC_ERROR during execution. (See test E24101A.)

- Array types.

  When an array type is declared with an index range exceeding the INTEGER'LAST values and with a component that is a null BOOLEAN array, this compiler raises NUMERIC_ERROR when the type is declared. (See tests E36202A and E36202B.)

A packed BOOLEAN array having a 'LENGTH exceeding INTEGER'LAST raises NUMERIC_ERROR when the array type is declared. (See test C52103X.)

A packed two-dimensional BOOLEAN array with more than INTEGER'LAST components raises NUMERIC_ERROR when the array type is declared. (See test C52104Y.)

A null array with one dimension of length greater than INTEGER'LAST may raise NUMERIC_ERROR either when declared or assigned. Alternatively, an implementation may accept the declaration. However, lengths must match in array slice assignments. This implementation raises NUMERIC_ERROR when the array type is declared. (See test E52103Y.)

In assigning one-dimensional array types, the entire expression appears to be evaluated before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype.

In assigning two-dimensional array types, the entire expression does not appear to be evaluated before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

. Discriminated types.

During compilation, an implementation is allowed to either accept or reject an incomplete type with discriminants that is used in an access type definition with a compatible discriminant constraint. This implementation rejects such subtype indications during compilation. (See test E38104A.)

In assigning record types with discriminants, the entire expression appears to be evaluated before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

. Aggregates.

In the evaluation of a multi-dimensional aggregate, all choices appear to be evaluated before checking against the index type. (See tests C43207A and C43207B.)

In the evaluation of an aggregate containing subaggregates, all choices are evaluated before being checked for identical bounds. (See test E43212B.)

All choices are evaluated before CONSTRAINT_ERROR is raised if a bound in a nonnull range of a nonnull aggregate does not belong to an index subtype. (See test E43211B.)

- Functions.

  The declaration of a parameterless function with the same profile as an enumeration literal in the same immediate scope is allowed by the implementation. (See test E66001D.)

- Representation clauses.

  Enumeration representation clauses are supported. (See test BC1002A.)

- Pragmas.

  The pragma INLINE is not supported for procedures nor is it supported for functions. (See tests CA3004E and CA3004F.)

- Input/output.

  The package SEQUENTIAL_IO cannot be instantiated with unconstrained array types and record types with discriminants. The package DIRECT_IO cannot be instantiated with unconstrained array types and record types with discriminants without defaults. (See tests CE2201D, CE2201E, and CE2401D.)

  More than one internal file can be associated with each external file for sequential I/O when both internal files are mode INFILE or when both internal files are mode OUTFILE. (See tests CE2107A..D and CE2107F (5 tests).)

  More than one internal file can be associated with each external file for direct I/O when both internal files are mode INFILE or when both internal files are mode OUTFILE. (See tests CE2107A..D and CE2107F (5 tests).)

  An external file associated with more than one internal file can be deleted. (See test CE2110B.)

  More than one internal file can be associated with each external file for text I/O for both reading and writing. (See tests CE3111A..E (5 tests).)

  An existing text file can be opened in OUT_FILE mode and can be created in both OUT_FILE and IN_FILE modes. (See test EE3102C.)

  Temporary sequential and direct files are not given a name. (See tests CE2108A and CE2108C.)

CHAPTER 3

TEST INFORMATION

3.1 TEST RESULTS

The AVF identified 2093 of the 2279 tests in Version 1.7 of the ACVC as potentially applicable to the validation of the C^3 Ada compiler, Version R00-00.00. Excluded were 170 tests requiring a floating-point precision greater than that supported by the implementation and the 16 withdrawn tests. After they were processed, 43 tests were determined to be inapplicable. The remaining 2050 tests were passed by the compiler.

The AVF concludes that the testing results demonstrate acceptable conformance to the Ada Standard.

3.2 SUMMARY OF TEST RESULTS BY CLASS

RESULT	TEST CLASS						TOTAL
	A	B	C	D	E	L	
Passed	66	816	1121	17	9	21	2050
Failed	0	0	0	0	0	0	0
Inapplicable	2	8	199	0	2	2	213
Withdrawn	0	4	12	0	0	0	16
TOTAL	68	828	1332	17	11	23	2279

3.3 SUMMARY OF TEST RESULTS BY CHAPTER

RESULT	CHAPTER												
	2	3	4	5	6	7	8	9	10	11	12	14	TOTAL
Passed	94	232	307	244	161	96	158	195	99	28	215	221	2050
Failed	0	0	0	0	0	0	0	0	0	0	0	0	0
Inapplicable	22	75	87	3	0	1	3	4	6	0	1	11	213
Withdrawn	0	1	4	0	0	0	1	2	6	0	1	1	16
TOTAL	116	308	398	247	161	97	162	201	111	28	217	233	2279

3.4 WITHDRAWN TESTS

The following tests have been withdrawn from the ACVC Version 1.7:

B4A010C	C41404A	CA1003B
B83A06B	C48008A	CA3005A through CA3005D (4 tests)
BA2001E	C4A014A	CE2107E
BC3204C	C92005A	
C35904A	C940ACA	

See Appendix D for the rationale for withdrawal.

3.5 INAPPLICABLE TESTS

Some tests do not apply to all compilers because they make use of features that a compiler is not required by the Ada Standard to support. Others may depend on the result of another test that is either inapplicable or withdrawn. For this validation attempt, 213 tests were inapplicable for the reasons indicated:

- C24113D..K (8 tests) have line lengths greater than MAX_IN_LEN.
- B37004A, B38105B, B74207A, BC3503A, and C48006B contain declarations that constrain incomplete types; this implementation rejects such declarations (see AI-00007).
- C34001E, B52004D, B55B09C, B86001CS, and C55B07A use LONG_INTEGER which is not supported by this compiler.
- C34001F, C35702A, and B86001CP use SHORT_FLOAT which is not supported by this compiler.

- C86001F redefines package SYSTEM, but TEXT_IO is made obsolete by this new definition in this implementation.

- C94004A..C (3 tests) require that library tasks continue execution after the main program terminates. But this implementation terminates library tasks when the main program terminates, as is allowed by AI-00399.

- C96005B checks implementations for which the smallest and largest values in type DURATION are different from the smallest and largest values in DURATION's base type. This is not the case for this implementation.

- CA3004E, EA3004C, and LA3004A use INLINE pragma for procedures which is not supported by this compiler.

- CA3004F, EA3004D, and LA3004B use INLINE pragma for functions which is not supported by this compiler.

- CE2107C, CE2107D, CE2108A, CE2108C, and CE3112A are inapplicable because temporary files do not have names.

- CE3111B is inapplicable because the TEXT_IO.PUT operation doesn't output to the external file until a subsequent NEW_LINE, RESET, or CLOSE operation.

- AE2101C, AE2101H, CE2201D, CE2201E, and CE2401D use instantiation of package SEQUENTIAL_IO with unconstrained array types which is not supported by this compiler.

- 170 tests were not processed because SYSTEM.MAX_DIGITS was 15. These tests were:

C24113L through C24113Y (14 tests)
C35705L through C35705Y (14 tests)
C35706L through C35706Y (14 tests)
C35707L through C35707Y (14 tests)
C35708L through C35708Y (14 tests)
C35802L through C35802Y (14 tests)
C45241L through C45241Y (14 tests)
C45321L through C45321Y (14 tests)
C45421L through C45421Y (14 tests)
C45424L through C45424Y (14 tests)
C45521L through C45521Z (15 tests)
C45621L through C45621Z (15 tests)

3.6 SPLIT TESTS

If one or more errors do not appear to have been detected in a Class B test because of compiler error recovery, then the test is split into a set of

smaller tests that contain the undetected errors. These splits are then compiled and examined. The splitting process continues until all errors are detected by the compiler or until there is exactly one error per split. Any Class A, Class C, or Class E test that cannot be compiled and executed because of its size is split into a set of smaller subtests that can be processed.

Splits were required for four Class B tests:

BC3204B BC3204D BC3205B BC3205D

3.7 ADDITIONAL TESTING INFORMATION

3.7.1 Prevalidation

Prior to validation, a set of test results for ACVC Version 1.7 produced by the C^3 Ada compiler, Version R00-00.00, running on the 3200 MPS computer was submitted to the AVF by the applicant for prevalidation review. Analysis of these results demonstrated that the compiler successfully passed all applicable tests.

3.7.2 Test Method

Testing of the C^3 Ada compiler using ACVC Version 1.7 was conducted on-site by a validation team. Two Concurrent Computer Corporation 3200 MPS computers operating under OS/32 running the Multi-Terminal Monitor (MTM) were used to run the full set of ACVC tests. In addition, the following Concurrent Computer Corporation computers were tested using a subset of the ACVC tests run on the 3200 MPS:

Host		Target	
Machine	Memory Size (megabytes)	Machine	Memory Size (megabytes)
3203	4	3203	4
3205	4	3205	4
3210	8	3210	8
3230	16	3230	16
3250	16	3250	16
3230 XP	8	3230 XP	8
3250 XP	16	3250 XP	16
3230 MPS	8	3230 MPS	8
3260 MPS	8	3260 MPS	8
3200 MPS	16	3280	8

This series of computers has identical architectures and operates under the OS/32 operating system.

Three magnetic tapes containing ACVC Version 1.7 were taken on-site by the validation team. The tapes contained all tests applicable to this validation, as well as all tests inapplicable to this validation, except for any Class C tests requiring a floating-point precision exceeding the maximum value supported by the implementation. Tests that make use of values that are specific to an implementation were customized before being written to the magnetic tapes.

The contents of the tapes were loaded to disk using a FORTRAN program developed by Concurrent Computer Corporation. After all the files were read to disk, Class B and C files and support files were backed up onto tape and loaded to disk on a second 3200 MPS computer. Class B tests BC3204B, BC3204D, and BC3205B were included in their split form on the tapes. Test BC3205D was edited into its split form on-site.

Tests were run using two identical 3200 MPS computers. One, designated MPSA, was dedicated to the validation testing and the second, MPSB, was used for other applications as well as validation testing. Two job streams were used for running tests on MPSA, and one job stream was used on MPSB. Support units, REPORT and CHECK_FILE, were compiled on each of the 3200 MPS computers and determined to be operating correctly. On MPSA, these units were compiled into a master program library that could be shared by the two job streams.

Tests were run in groups according to test class. For the large classes, B and C, tests were run in smaller groups according to chapter within class. The program library for each job stream was cleared to reclaim disk space after each group of tests had been run. The list files for each group of tests were compressed to save disk space. Upon completion of the group of tests, the files were expanded and printed.

In parallel with the full validation on the 3200 MPS computers, a subset of the ACVC Version 1.7 was run on other computers in the same series--i.e., 3203, 3205, 3210, 3230, 3250, 3230 XP, 3250 XP, 3230 MPS, and 3260 MPS. The subset of 60 tests consisted of five tests selected at random from the classes of tests within each chapter. The 60 tests were backed up onto tape from the 3200 MPS and loaded to disk for each computer. The tests were compiled, linked, and executed (as appropriate) on the individual computers. Test results were printed from each computer and reviewed by the validation team.

A cross-compiler was tested for a 3200 MPS host computer and a 3280 target computer. A subset of the ACVC Version 1.7 tests was compiled and linked on the 3200 MPS. The executable images were backed up onto tape and loaded to disk on the 3280. Tests were executed, and results were printed from the target computer. Results were identical to those generated by the compiler operating on the 3200 MPS.

TEST INFORMATION

The compiler for the Concurrent Computer Corporation Series 3200 was tested using command scripts provided by Concurrent Computer Corporation. These scripts were reviewed by the validation team. The following options were in effect for testing:

LIST — The LIST option controls the generation of the source listing from the compiler. A listing of all source lines is generated.

OPTIMIZE — This option controls the action of performing simple optimizations like constant folding, dead code elimination, and peephole optimization.

PAGE_SIZE — This option specifies the number of significant lines per page on the listing file. The default is 60 lines per page.

SEGMENTED — This option specifies that the code generated is to be segmented in PURE and IMPURE code.

Tests were run in batch mode for each of the configurations tested. When testing was completed, test output from the executable tests, compilation listings for Class B tests, job log files, and the compiler and its environment were captured on magnetic tapes. These tapes were archived at the AVF. The listings examined on-site by the validation team were also archived.

3.7.3 Test Site

The validation team arrived at Concurrent Computer Corporation, Tinton Falls NJ on 4 JUN 1986 and departed after testing was completed on 11 JUN 1986.

APPENDIX A

COMPLIANCE STATEMENT

Concurrent Computer Corporation has submitted the following compliance statement concerning the C^3 Ada compiler.

Compliance Statement

Base Configuration:

Compiler: C^3 Ada, Version R00-00.00

Test Suite: Ada Compiler Validation Capability, Version 1.7

Host Computer:

Machine(s): Concurrent Computer Corporation Series 3200

Operating System: OS/32, Version R08-01

Target Computer:

Machine(s): Concurrent Computer Corporation Series 3200

Operating System: OS/32, Version R08-01

Concurrent Computer Corporation has made no deliberate extensions to the Ada language standard.

Concurrent Computer Corporation agrees to the public disclosure of this report.

Concurrent Computer Corporation agrees to comply with the Ada trademark policy, as defined by the Ada Joint Program Office.

Date: 6/20/86

Concurrent Computer Corporation
Seetharama Shastry
Manager of Software Development

APPENDIX B

APPENDIX F OF THE Ada STANDARD

The only allowed implementation dependencies correspond to implementation-dependent pragmas, to certain machine-dependent conventions as mentioned in chapter 13 of MIL-STD-1815A, and to certain allowed restrictions on representation classes. The implementation-dependent characteristics of the C^3 Ada compiler, Version R00-00.00, are described in the following sections which discuss topics one through eight as stated in Appendix F of the Ada Language Reference Manual (ANSI/MIL-STD-1815A). Package STANDARD is also included in this appendix.

Appendix F

IMPLEMENTATION-DEPENDENT CHARACTERISTICS

F.1 Introduction

The following sections provide all implementation-dependent characteristics of the C^3 Ada compiler.

F.2 Implementation-Dependent Pragmas

The following is the syntax representation of a pragma:

```
pragma IDENTIFIER [(ARGUMENT {,ARGUMENT})];
```

where:

IDENTIFIER	is the name of the pragma.
ARGUMENT	defines a parameter of the pragma. For example, the LIST pragma expects the arguments ON or OFF.

The following table summarizes all of the recognized pragmas and whether they are implemented or not.

TABLE F-1. SUMMARY OF RECOGNIZED PRAGMAS

PRAGMA	Implemented	Comments
CONTROLLED	No	automatic storage reclamation of unreferenced access objects is not applicable to the C^3 Ada implementation.
ELABORATE	Yes	is handled as defined by the Ada language.
INLINE	No	Subprogram bodies are not expanded inline at each call.
INTERFACE	Yes	is implemented for ASSEMBLER and FORTRAN.
LIST	Yes	is handled as defined by the Ada language.
MEMORY_SIZE	No	The user cannot specify the number of available storage units in the machine configuration which is defined in package SYSTEM.
OPTIMIZE	No	The user cannot specify either time or space as the primary optimization criterion.
PACK	Yes	the elements of an array or record are packed down to a

PRAGMA	Implemented	Comments
		minimal number of bytes.
PAGE	Yes	is handled as defined by the Ada language.
PRIORITY	No	The task or main_program can not have priority.
SHARED	No	Not applicable because every read or update of the variable declared by an object declaration and whose type is a scaler or access type is a synchronization point for that variable.
STORAGE_UNIT	No	The user can not specify the number of bits per storage unit, which is defined in package SYSTEM.
SUPPRESS	No	All run-time checks, such as ACCESS_CHECK, INDEX_CHECK, RANGE_CHECK, etc. cannot be suppressed for any specific type, object, subprogram etc.. See the description of SUPPRESS_ALL.
SYSTEM_NAME	No	The user can not specify the target system name, which is defined in package SYSTEM.
SUPPRESS_ALL	Yes	This pragma gives the compiler permission to omit all of the following run-time checks for all types and objects in the designated compilation units; ACCESS_CHECK, DISCRIMINANT_CHECK, INDEX_CHECK, RANGE_CHECK, LENGTH_CHECK, and OVER_FLOW_CHECK for all integer calculations. SUPPRESS_ALL can occur anywhere that the pragma SUPPRESS is allowed. See the *Ada Language Reference Manual* for the placement of pragma SUPPRESS.

F.3 Length Clauses

A length clause specifies the amount of storage associated with a given type. The following is a list of the implementation-dependent attributes.

T'SIZE must be a multiple of eight. Must be 32 for a type derived from FLOAT, and 64 for a type derived from LONG_FLOAT. For array and record types, only the size chosen by the compiler may be specified.

T'STORAGE_SIZE is fully supported for collection size specification.

T'STORAGE_SIZE is not supported for task activation. Task memory is limited by the work space for the program.

T'SMALL must be a power of two for a fixed point type.

F.4 Representation Attributes

The Representation attributes listed below are as described in the *Ada Language Reference Manual*, section 13.7.2.

X'ADDRESS
X'SIZE
R.C'POSITION
R.C'FIRST_BIT
R.C'LAST_BIT

T'STORAGE_SIZE for access types, returns the current amount of storage reserved for the type. If a T'STORAGE_SIZE representation clause has been specified, then the amount specified is returned. Otherwise the current amount allocated is returned.

T'STORAGE_SIZE for task types or objects is not implemented. It returns 0.

F.4.1 Representation Attributes of Real Types

P'DIGITS yields the number of decimal digits for the subtype P. This value is 6 for type FLOAT, and 15 for type LONG_FLOAT.

P'MANTISSA yields the number of binary digits in the mantissa of P. The value is 21 for type FLOAT, and 51 for type LONG_FLOAT.

DIGITS	MANTISSA	DIGITS	MANTISSA	DIGITS	MANTISSA
1	5	6	21	11	38
2	8	7	25	12	41
3	11	8	28	13	45
4	15	9	31	14	48
5	18	10	35	15	51

P'EMAX yields the largest exponent value of model numbers for the subtype P. The value is 84 for type FLOAT, and 204 for type LONG_FLOAT.

DIGITS	EMAX	DIGITS	EMAX	DIGITS	EMAX
1	20	6	84	11	152
2	32	7	100	12	164
3	60	8	112	13	180
4	72	9	124	14	192
5	84	10	140	15	204

P'EPSILON yields the absolute value of the difference between the model number 1.0 and the next model number above for the subtype P. The value is 16#0.00001# for type FLOAT, and 16#0.0000_0000_0000_4# for type LONG_FLOAT.

DIGITS	EPSILON	DIGITS	EPSILON	DIGITS	EPSILON
1	16#0.1#E00	6	16#0.1#E-4	11	16#0.8#E-9
2	16#0.2#E-1	7	16#0.1#E-5	12	16#0.1#E-9
3	16#0.4#E-2	8	16#0.2#E-6	13	16#0.1#E-10
4	16#0.4#E-3	9	16#0.4#E-7	14	16#0.2#E-11
5	16#0.8#E-4	10	16#0.4#E-8	15	16#0.4#E-12

P'SMALL yields the smallest positive model number of the subtype P. The value is 16#0.8#E-21 for type FLOAT, and 16#0.8#E-51 for type LONG_FLOAT.

DIGITS	SMALL	DIGITS	SMALL	DIGITS	SMALL
1	16#0.8#E-5	6	16#0.8#E-21	11	16#0.8#E-38
2	16#0.8#E-8	7	16#0.8#E-25	12	16#0.8#E-41
3	16#0.8#E-11	8	16#0.8#E-28	13	16#0.8#E-45
4	16#0.8#E-15	9	16#0.8#E-31	14	16#0.8#E-48
5	16#0.8#E-18	10	16#0.8#E-35	15	16#0.8#E-51

P'LARGE yields the largest positive model number of the subtype P. The value is 16#0.FFFFFF8#E21 for type FLOAT, and 16#0.FFFF_FFFF_FFFF_E#E51 for type LONG_FLOAT.

DIGITS	LARGE
1	16#0.F8#E5
2	16#0.FF#E8
3	16#0.FFE#E11
4	16#0.FFFE#E15
5	16#0.FFFF_C#E18
6	16#0.FFFF_F8#E21
7	16#0.FFFF_FF8#E25
8	16#0.FFFF_FFF#E28
9	16#0.FFFF_FFFE#E31
10	16#0.FFFF_FFFF_E#E35
11	16#0.FFFF_FFFF_FC#E38
12	16#0.FFFF_FFFF_FF8#E41
13	16#0.FFFF_FFFF_FFF8#E45
14	16#0.FFFF_FFFF_FFFF#E48
15	16#0.FFFF_FFFF_FFFF_E#E51

P'SAFE_EMAX Yields the largest exponent value of safe numbers of type T. The value is 252 for types FLOAT and LONG_FLOAT.

P'SAFE_SMALL Yields the smallest positive safe number of type T. The value is 16#0.1#E-64 for types FLOAT and LONG_FLOAT.

P'SAFE_LARGE Yields the largest positive safe number of the type P. The value is 16#0.FFFF_FF#E63 for type FLOAT, and 16#0.FFFF_FFFF_FFFF_FF#E63 for type LONG_FLOAT.

P'MACHINE_ROUNDS is true.

P'MACHINE_OVERFLOWS is true.

P'MACHINE_RADIX is 16.

P'MACHINE_MANTISSA is 6 for types derived from FLOAT; else 14.

P'MACHINE_EMAX is 63.
P'MACHINE_EMIN is -64.

F.4.2 Representation Attributes of Fixed Point Types

For any Fixed Point Type T, the representation attributes are:

T'MACHINE_ROUNDS true
T'MACHINE_OVERFLOWS true

F.4.3 Enumeration Representation Clauses

The maximum number of elements in an enumeration type is 2,147,483,647.

RESTRICTIONS - None.

F.4.4 Record Representation Clauses

The *Ada Language Reference Manual* states that an implementation may generate names that denote implementation-dependent components. This is not present in this release of the C^3 Ada compiler.

RESTRICTIONS - Components must be placed at a storage position that is a multiple of eight. Floating-point types must be fullword aligned, that is, placed at a storage position that is a multiple of 32.

Record components of a private type can not be included in a record representation specification.

F.4.5 Type Duration

Duration'small equals 0.06103515625 milliseconds or 2^{-14} seconds. This number is the smallest power of 2 which can still represent the number of seconds in a day in a fullword fixed-point number.

System.tick equals 100 milliseconds. The actual computer clock-tick is 1.0/120.0 seconds (or about 8.33333 milliseconds) in 60 HZ areas and 1.0/100.0 seconds (or 10 milliseconds) in 50HZ areas. System.tick represents the lowest common numerator, which can be integrally divided by the actual clock-tick from both areas.

Duration'small is significantly smaller than the actual computer clock-tick. Therefore, the least amount of delay possible is limited by the actual clock-tick. The delay of duration'small follows this formula:

<actual-clock-tick> ± <actual-clock-tick> + 4.45ms

The 4.45ms represents the overhead or the minimum delay possible on a 3250 or 3200MPS CPU. For 60 HZ areas, the range of delay is approximately from 4.45ms to 21.11666ms. For 50 HZ areas, the range of delay is approximately from 4.45ms to 24.45ms. However, on the average, the delay is slightly greater than the actual clock-tick.

In general, the formula for finding the range of a delay value, *x*, is:

nearest_multiple(x,<actual-clock-tick>) ± <actual-clock-tick> + 4.45ms

where nearest_multiple rounds x up to the nearest multiple of the actual clock-tick.

TABLE F-2. TYPE DURATION

DURATION'DELTA	2#1.0#E-14	$\simeq$ 61 μs
DURATION'SMALL	2#1.0#E-14	$\simeq$ 61 μs
DURATION'FIRST	-131072.00	$\simeq$ 36 hrs
DURATION'LAST	131071.99993896484375	$\simeq$ 36 hrs

F.5 Address Clauses

Address clauses are implemented for objects. No storage is allocated for objects with address clauses by the compiler. The user must guarantee the storage for these by some other means (e.g. through the use of the Absolute Instruction found in the *CAL/32 Reference Manual*). The exception PROGRAM_ERROR is raised upon reference to the object if the specified address is not in the program's address space or is not properly aligned.

RESTRICTIONS - Address clauses are not implemented for subprograms, packages, or task units. In addition, address clauses are not available for use with task entries (i.e. interrupts).

F.6 The Package SYSTEM

The package SYSTEM, provided with C^3 Ada, permits access to machine dependent features. The specification of the package SYSTEM declares constant values dependent on the Series 3200 systems. The following is a listing of the visible section of the package SYSTEM specification.

```
package SYSTEM is

    type ADDRESS is private;
    type NAME    is (CCUR_3200);


    SYSTEM_NAME    :  constant NAME := CCUR_3200;
    STORAGE_UNIT   :  constant := 8;
    MEMORY_SIZE    :  constant := 2 ** 24;
    MIN_INT        :  constant := -2_147_483_648;
    MAX_INT        :  constant := 2_147_483_647;
    MIN_DIGITS     :  constant := 15;
    MAX_MANTISSA   :  constant := 31;
    FINE_DELTA     :  constant := 2#1.0#E-30;
    TICK           :  constant := 0.1;



    type UNSIGNED_SHORT_INTEGER is range    0 .. 65_535;
    type UNSIGNED_TINY_INTEGER is range     0 ..    255;


    for UNSIGNED_SHORT_INTEGER'SIZE use 16;
    for UNSIGNED_TINY_INTEGER'SIZE use 8;


    subtype PRIORITY          is INTEGER range 0 .. 255;  --ignored
    subtype BYTE              is UNSIGNED_TINY_INTEGER;
    subtype ADDRESS_RANGE     is INTEGER range 0 .. 2**24-1;


    function INTEGER_TO_ADDRESS (ADDR : ADDRESS_RANGE)    return ADDRESS;
    function ADDRESS_TO_INTEGER (ADDR : ADDRESS)    return ADDRESS_RANGE;


    function "+" (ADDR : ADDRESS; OFFSET : INTEGER) return ADDRESS;
    function "-" (ADDR : ADDRESS; OFFSET : INTEGER) return ADDRESS;

private
  -- implementation dependent
end SYSTEM;
```

F.7 Interface to Other Languages

Pragma INTERFACE is implemented for two languages, assembler and FORTRAN. The pragma can take one of three forms:

1. **For any assembly language procedure or function:**

```
pragma INTERFACE (ASSEMBLER, ROUTINE_NAME);
```

2. **For FORTRAN procedures or functions with only IN parameters:**

```
pragma INTERFACE (FORTRAN, ROUTINE_NAME);
```

3. For FORTRAN functions that have IN OUT or OUT parameters:

```
pragma INTERFACE (FORTRAN, ROUTINE_NAME, IS_FUNCTION);
```

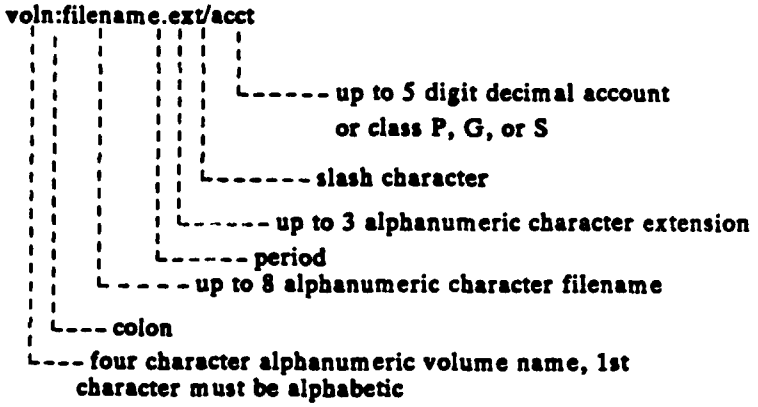
In C^3 Ada, functions may not have IN OUT or OUT parameters so the Ada specification for the function is written as a procedure with the first argument being the function return result. Then, the parameter "is_function" is specified to inform the compiler that it is, in reality, a FORTRAN function.

F.8 Input-Output Packages

The following two system dependent parameters are used for the control of external files:

- NAME parameter
- FORM parameter

The NAME parameter must be an OS/32 file name string. OS/32 filenames are specified as follows:



The implementation-dependent values used for keywords in the FORM parameter are discussed below. An example of the FORM parameter with possible values specified can be found in the section on procedure CREATE (2.4.1) and procedure OPEN (2.4.2) in the *C^3 Ada Run Time Reference Manual.*

lu an integer in the range 0..254 specifying the logical unit number.

fo specifies legal OS/32 file formats (file organization). They are:

INDEX | IN
CONTIGUOUS | CO
NON_BUFFERED | NB
EXTENDIBLE_CONTIGUOUS | EXTENDABLE_CONTIGUOUS | EC
LONG_RECORD | LR
ITAM
DEVICE

rs an integer in the range 1..65535 specifying the physical record size.

1. For INDEX, ITAM (Inter Telecommunications Access Method) and NON_BUFFERED files, this specifies the physical record size.
2. The physical record size for CONTIGUOUS and EXTENDIBLE_CONTIGUOUS files is determined by rounding the element size up to the nearest 256-byte boundary. For such files, *rs* is ignored.

3. The physical record size for LONG_RECORD files is specified by the data blocking factor multiplied by 256 and *rs* is ignored.

4. For a DEVICE the physical record size always equals the element size and *rs* is ignored.

dbf Data_blocking_factor. An integer in the range 0..255 (as set up at OS/32 sysgen time) that specifies the number of contiguous disk sectors (256 bytes) in a data block. It applies only to INDEX, NON_BUFFERED, EXTENDIBLE_CONTIGUOUS and LONG_RECORD files. For other file organizations (see *file_organization* above), it is ignored. A value of 0 causes the data blocking factor to be set to the current OS/32 default.

ibf Index_blocking_factor. An integer in the range 0..255 (as set up at OS/32 sysgen time) specifying the number of contiguous disk sectors (256 bytes) in an index block of an INDEX, NON_BUFFERED, EXTENDIBLE_CONTIGUOUS, or LONG_RECORD file. For other file organizations (see *file_organization* above), it is ignored.

al Allocation. An integer in the range 1..2,147,483,647. For CONTIGUOUS files, it specifies the number of 256 byte sectors. For ITAM files, it specifies the physical block size in bytes associated with the buffered terminal. For other file organizations, (see *file_organization* above), it is ignored.

pr Privileges. Specifies OS/32 access privileges, e.g., SRO, ERO, SWO, EWO, SRW, SREW, ERSW, and ERW.

keys READ/WRITE keys. A decimal or hexadecimal integer specifying the OS/32 READ/WRITE keys, which range from 16#0000# to 16#FFFF#(0..65535). The left two hexadecimal digits signify the write protection key and the right two hexadecimal digits signify the read protection key. For more information on protection keys see the *OS/32 MTM primer.*

pad Pad character. Specifies the padding character used for READ and WRITE operations; the pad character is either NONE, BLANK, or NUL. The default is NONE.

TABLE F-3. PAD CHARACTER OPTIONS

Pad Character	Action
NONE	Records are not padded. (Default.)
NUL	Records are padded with ASCII.NUL.
BLANK	Records are padded with blanks and OS/32 ASCII I/O operations are used.

dc Device code. An integer in the range 0..255 specifying the OS/32 device code of the external file. See the *OS/32 System Generation Reference Manual* for a list of all devices and their respective codes.

da Device attributes. An integer in the range 0..65535 specifying the OS/32 device attributes of the external file. See the *OS/32 SVC Reference Manual* (Chapter 7, the table entitled Description and Mask Values of the Device Attributes Field). for all devices and their respective attributes.

ds Device status. An integer in the range 0..65535 specifying the status of the external file. A status of 0 means that the access to the file terminated with no errors;

otherwise a device error has occurred. For errors occurring during READ and WRITE operations, the status values and their meanings are found in Chapter 2 (The tables on Device-Independent and Device-Dependent Status Codes) of the *OS/32 SVC Reference Manual.*

ps — Prompting string. This quoted string is output on the terminal before the GET operation only if the file is associated with a terminal; otherwise this FORM parameter is ignored. The default is the null string, in which case no string is output to the terminal.

character_io — If character_io is specified in the FORM string, the only other allowable FORM parameters are LU => *lu*, FILE_ORGANIZATION => DEVICE, PRIVILEGE=> SRW. Furthermore, the NAME string must denote a terminal or interactive device. In order for character_io to work properly, the user must specify ENABLE TYPEAHEAD to MTM, to turn on BIOC's type ahead feature.

F.8.1 TEXT I/O

There are two implementation dependent types for TEXT_IO: COUNT and FIELD. Their declarations implemented for the C^3 Ada compiler are as follows:

```
type COUNT is range 0 ..INTEGER'LAST;
subtype FIELD is INTEGER range 0 ..255;
```

F.8.1.1 End-of-file Markers

When working with text files the following representations are used for end-of-file markers. A line terminator followed by a page terminator is represented by:

```
ASCII.FF  ASCII.CR
```

A line terminator followed by a page terminator, which is then followed by a file terminator is represented by:

```
ASCII.FF ASCII.EOT ASCII.CR
```

End-of-file may also be represented as the physical end-of-file. For input from a terminal, the combination above is represented by the control characters:

```
ASCII.FF ASCII.EOT ASCII.CR
```

or with BIOC: ASCII.DC4 ASCII.EOT ASCII.CR, i.e., ^T ^D <cr>

F.8.2 Restrictions on ELEMENT_TYPE

The following are the restrictions concerning ELEMENT_TYPE:

1. Input/Output of objects that contain "pointers" is allowed but the fundamental association between the access variables and its accessed type is ignored.
2. The maximum size of variant data type is always used.

3. If the number of bytes output is less than the physical record length, the rest of the record is padded with zeros, when PAD => NUL is used; blanks when PAD => BLANK is used; and otherwise no padding is performed.
4. If the number of bytes input is less than the logical record length, the rest of the record is padded with zeros, when PAD => NUL is used; blanks, when PAD => BLANK is used; and otherwise is not changed.
5. If the size of the element is exceeded by the physical record size on a READ operation, the extra data in the physical record is lost. DATA_ERROR is not raised.
6. If the size of the element exceeds the physical record size on a WRITE operation, the extra data in the element is lost. DATA_ERROR is not raised.
7. ELEMENT_TYPE can not be unconstrained for SEQUENTIAL_IO and DIRECT_IO.

F.9 Unchecked Programming

Unchecked programming gives the programmer the ability to circumvent some of the strong typing and elaboration rules of the Ada Language. As such, it is the programmer's responsibility to ensure that the guidelines provided in the following sections are followed.

F.9.1 Unchecked Storage Deallocation

The unchecked storage deallocation generic procedure explicitly deallocates the space for a dynamically acquired object.

RESTRICTIONS - This procedure frees storage only if:

1. The object being deallocated was the last one allocated of all objects in a given declarative part.
2. All objects in a single chunk of the collection belonging to all access types declared in the same declarative part are deallocated.

F.9.2 Unchecked Type Conversions

The unchecked type conversion generic function permits the user to convert, without type checking, from one type to another. It is the user's responsibility to guarantee that such a conversion preserves the properties of the target type.

RESTRICTIONS - The object used as the parameter in the function may not:

1. Be an access or task type.
2. Contain components of access or task types.
3. Have components which contain dynamic or unconstrained array types.

If the target'size is greater than the source'size, the resulting conversion is unpredictable. If the target'size is less than the source'size, the result is that the leftmost bits of the source are placed in the target.

Since unchecked_conversion is implemented as an arbitrary block move, no alignment constraints are necessary on the source or the target operands.

F.10 Implementation-Dependent Restrictions

1. The main procedure must be parameterless.
2. The source line length must be less than or equal to 80 characters.
3. Due to the source line length, the largest identifier is 80 characters.
4. The maximum number of library units is 1000.
5. The maximum number of bits in an object is 2^{31} -1.
6. The maximum static nesting level is 63.
7. The maximum number of directly imported units of a single compilation unit must not exceed 63.

This annex outlines the specification of the package STANDARD containing all predefined identifiers in the language. The corresponding package body is implementation-defined and is not shown.

The operators that are predefined for the types declared in the package STANDARD are given in comments since they are implicitly declared. Italics are used for pseudo-names of anonymous types (such as *universal_real*) and for undefined information (such as *implementation_defined* and *any_fixed_point_type*).

```
package STANDARD is

   type BOOLEAN is (FALSE, TRUE);

   -- The predefined relational operators for this type are as follows:

   -- function "="    (LEFT, RIGHT : BOOLEAN) return BOOLEAN;
   -- function "/="   (LEFT, RIGHT : BOOLEAN) return BOOLEAN;
   -- function "<"    (LEFT, RIGHT : BOOLEAN) return BOOLEAN;
   -- function "<="   (LEFT, RIGHT : BOOLEAN) return BOOLEAN;
   -- function ">"    (LEFT, RIGHT : BOOLEAN) return BOOLEAN;
   -- function ">="   (LEFT, RIGHT : BOOLEAN) return BOOLEAN;
```

```
-- The predefined logical operators and the predefined logical negation operator are as follows:

-- function "and"  (LEFT, RIGHT : BOOLEAN) return BOOLEAN;
-- function "or"   (LEFT, RIGHT : BOOLEAN) return BOOLEAN;
-- function "xor"  (LEFT, RIGHT : BOOLEAN) return BOOLEAN;

-- function "not"  (RIGHT : BOOLEAN) return BOOLEAN;


-- The universal type universal_integer is predefined.

type INTEGER is range -2_147_483_648..2_147_147_483_647;

-- The predefined operators for this type are as follows:

-- function "="    (LEFT, RIGHT : INTEGER) return BOOLEAN;
-- function "/="   (LEFT, RIGHT : INTEGER) return BOOLEAN;
-- function "<"    (LEFT, RIGHT : INTEGER) return BOOLEAN;
-- function "<="   (LEFT, RIGHT : INTEGER) return BOOLEAN;
-- function ">"    (LEFT, RIGHT : INTEGER) return BOOLEAN;
-- function ">="   (LEFT, RIGHT : INTEGER) return BOOLEAN;
-- function "+"    (RIGHT : INTEGER) return INTEGER;
-- function "-"    (RIGHT : INTEGER) return INTEGER;
-- function "abs"  (RIGHT : INTEGER) return INTEGER;

-- function "+"    (LEFT, RIGHT : INTEGER) return INTEGER;
-- function "-"    (LEFT, RIGHT : INTEGER) return INTEGER;
-- function "*"    (LEFT, RIGHT : INTEGER) return INTEGER;
-- function "/"    (LEFT, RIGHT : INTEGER) return INTEGER;
-- function "rem"  (LEFT, RIGHT : INTEGER) return INTEGER;
-- function "mod"  (LEFT, RIGHT : INTEGER) return INTEGER;

-- function "**"   (LEFT : INTEGER; RIGHT : INTEGER) return INTEGER;

-- An implementation may provide additional predefined integer types. It is recommended that the
-- names of such additional types end with INTEGER as in SHORT_INTEGER or LONG_INTEGER.
-- The specification of each operator for the type universal_integer, or for any additional
-- predefined integer type, is obtained by replacing INTEGER by the name of the type in the
-- specification of the corresponding operator of the type INTEGER, except for the right operand
-- of the exponentiating operator.


-- The universal type universal_real is predefined.

type FLOAT is digits 6 range -16*0.FFFFFF*E63..16*0.FFFFFF*E63;

-- The predefined operators for this type are as follows:

-- function "="    (LEFT, RIGHT : FLOAT) return BOOLEAN;
-- function "/="   (LEFT, RIGHT : FLOAT) return BOOLEAN;
-- function "<"    (LEFT, RIGHT : FLOAT) return BOOLEAN;
-- function "<="   (LEFT, RIGHT : FLOAT) return BOOLEAN;
-- function ">"    (LEFT, RIGHT : FLOAT) return BOOLEAN;
-- function ">="   (LEFT, RIGHT : FLOAT) return BOOLEAN;
```

```
-- function "+"   (RIGHT : FLOAT) return FLOAT;
-- function "-"   (RIGHT : FLOAT) return FLOAT;
-- function "abs" (RIGHT : FLOAT) return FLOAT;

-- function "+"   (LEFT, RIGHT : FLOAT) return FLOAT;
-- function "-"   (LEFT, RIGHT : FLOAT) return FLOAT;
-- function "*"   (LEFT, RIGHT : FLOAT) return FLOAT;
-- function "/"   (LEFT, RIGHT : FLOAT) return FLOAT;

-- function "**"  (LEFT : FLOAT; RIGHT : INTEGER) return FLOAT;

-- An implementation may provide additional predefined floating point types. It is recom-
-- mended that the names of such additional types end with FLOAT as in SHORT_FLOAT or
-- LONG_FLOAT. The specification of each operator for the type universal_real, or for any
-- additional predefined floating point type, is obtained by replacing FLOAT by the name of the
-- type in the specification of the corresponding operator of the type FLOAT.

-- In addition, the following operators are predefined for universal types:

-- function "*" (LEFT : universal_integer; RIGHT : universal_real)    return universal_real;
-- function "*" (LEFT : universal_real;    RIGHT : universal_integer) return universal_real;
-- function "/" (LEFT : universal_real;    RIGHT : universal_integer) return universal_real;

-- The type universal_fixed is predefined. The only operators declared for this type are

-- function "*" (LEFT : any_fixed_point_type; RIGHT : any_fixed_point_type) return universal_fixed;
-- function "/" (LEFT : any_fixed_point_type; RIGHT : any_fixed_point_type) return universal_fixed;


-- The following characters form the standard ASCII character set. Character literals cor-
-- responding to control characters are not identifiers; they are indicated in italics in this definition.


type CHARACTER is

    ( nul,  soh,  stx,  etx,    eot,  enq,  ack,  bel,
      bs,   ht,   lf,   vt,     ff,   cr,   so,   si,
      dle,  dc1,  dc2,  dc3,    dc4,  nak,  syn,  etb,
      can,  em,   sub,  esc,    fs,   gs,   rs,   us,

      ' ',  '!',  '"',  '#',    '$',  '%',  '&',  ''',
      '(',  ')',  '*',  '+',    ',',  '-',  '.',  '/',
      '0',  '1',  '2',  '3',    '4',  '5',  '6',  '7',
      '8',  '9',  ':',  ';',    '<',  '=',  '>',  '?',

      '@',  'A',  'B',  'C',    'D',  'E',  'F',  'G',
      'H',  'I',  'J',  'K',    'L',  'M',  'N',  'O',
      'P',  'Q',  'R',  'S',    'T',  'U',  'V',  'W',
      'X',  'Y',  'Z',  '[',    '\',  ']',  '^',  '_',

      '`',  'a',  'b',  'c',    'd',  'e',  'f',  'g',
      'h',  'i',  'j',  'k',    'l',  'm',  'n',  'o',
      'p',  'q',  'r',  's',    't',  'u',  'v',  'w',
      'x',  'y',  'z',  '{',    '|',  '}',  '~',  del );
```

```
for CHARACTER use -- 128 ASCII character set without holes
    (0, 1, 2, 3, 4, 5, ..., 125, 126, 127);

-- The predefined operators for the type CHARACTER are the same as for any enumeration type

   type SHORT_INTEGER is range -32768..32767;

   -- The predefined operators for this type are as follows:

   -- function "="     (LEFT, RIGHT : SHORT_INTEGER) return BOOLEAN;
   -- function "/="    (LEFT, RIGHT : SHORT_INTEGER) return BOOLEAN;
   -- function "<"     (LEFT, RIGHT : SHORT_INTEGER) return BOOLEAN;
   -- function "<="    (LEFT, RIGHT : SHORT_INTEGER) return BOOLEAN;
   -- function ">"     (LEFT, RIGHT : SHORT_INTEGER) return BOOLEAN;
   -- function ">="    (LEFT, RIGHT : SHORT_INTEGER) return BOOLEAN;

   -- function "+"     (RIGHT : SHORT_INTEGER) return SHORT_INTEGER;
   -- function "-"     (RIGHT : SHORT_INTEGER) return SHORT_INTEGER;
   -- function "abs"   (RIGHT ; SHORT_INTEGER) return SHORT_INTEGER;

   -- function "+"     (LEFT, RIGHT : SHORT_INTEGER) return SHORT_INTEGER;
   -- function "-"     (LEFT, RIGHT : SHORT_INTEGER) return SHORT_INTEGER;
   -- function "*"     (LEFT, RIGHT : SHORT_INTEGER) return SHORT_INTEGER;
   -- function "/"     (LEFT, RIGHT : SHORT_INTEGER) return SHORT_INTEGER;
   -- function "rem"   (LEFT, RIGHT : SHORT_INTEGER) return SHORT_INTEGER;
   -- function "mod"   (LEFT, RIGHT : SHORT_INTEGER) return SHORT_INTEGER;

   -- function "**"    (LEFT : SHORT_INTEGER; RIGHT : INTEGER) return SHORT_INTEGER;


   type TINY_INTEGER is range  -128..127;

   -- The predefined operators for this type are as follows:

   -- function "="     (LEFT, RIGHT : TINY_INTEGER) return BOOLEAN;
   -- function "/="    (LEFT, RIGHT : TINY_INTEGER) return BOOLEAN;
   -- function "<"     (LEFT, RIGHT : TINY_INTEGER) return BOOLEAN;
   -- function "<="    (LEFT, RIGHT : TINY_INTEGER) return BOOLEAN;
   -- function ">"     (LEFT, RIGHT : TINY_INTEGER) return BOOLEAN;
   -- function ">="    (LEFT, RIGHT : TINY_INTEGER) return BOOLEAN;

   -- function "+"     (RIGHT : TINY_INTEGER) return TINY_INTEGER;
   -- function "-"     (RIGHT : TINY_INTEGER) return TINY_INTEGER;
   -- function "abs"   (RIGHT : TINY_INTEGER) return TINY_INTEGER;
```

```
   -- function "+"     (LEFT, RIGHT : TINY_INTEGER) return TINY_INTEGER;
   -- function "-"     (LEFT, RIGHT : TINY_INTEGER) return TINY_INTEGER;
   -- function "*"     (LEFT, RIGHT : TINY_INTEGER) return TINY_INTEGER;
   -- function "/"     (LEFT, RIGHT : TINY_INTEGER) return TINY_INTEGER;
   -- function "rem"   (LEFT, RIGHT : TINY_INTEGER) return TINY_INTEGER;
   -- function "mod"   (LEFT, RIGHT : TINY_INTEGER) return TINY_INTEGER;

   -- function "**"    (LEFT : TINY_INTEGER; RIGHT : INTEGER) return TINY_INTEGER;

   type LONG_FLOAT is digits 15 range -16#0.FF_FFFF_FFFF_FFFF#E63..
                                       16#0.FF_FFFF_FFFF_FFFF#E63;


   -- function "="     (LEFT, RIGHT : LONG_FLOAT)return BOOLEAN;
   -- function "/="    (LEFT, RIGHT : LONG_FLOAT)return BOOLEAN;
   -- function "<"     (LEFT, RIGHT : LONG_FLOAT)return BOOLEAN;
   -- function "<="    (LEFT, RIGHT : LONG_FLOAT)return BOOLEAN;
   -- function ">"     (LEFT, RIGHT : LONG_FLOAT)return BOOLEAN;
   -- function ">="    (LEFT, RIGHT : LONG_FLOAT)return BOOLEAN;

   -- function "+"     (RIGHT : LONG_FLOAT) return LONG_FLOAT;
   -- function "-"     (RIGHT : LONG_FLOAT) return LONG_FLOAT;
   -- function "abs"   (RIGHT : LONG_FLOAT) return LONG_FLOAT;

   -- function "+"     (LEFT, RIGHT : LONG_FLOAT) return LONG_FLOAT;
   -- function "-"     (LEFT, RIGHT : LONG_FLOAT) return LONG_FLOAT;
   -- function "*"     (LEFT, RIGHT : LONG_FLOAT) return LONG_FLOAT;
   -- function "/"     (LEFT, RIGHT : LONG_FLOAT) return LONG_FLOAT;

   -- function "**"    (LEFT : LONG_FLOAT; RIGHT : INTEGER) return LONG_FLOAT;

package ASCII is

   -- Control characters:

   NUL  : constant CHARACTER := nul;    SOH  : constant CHARACTER := soh;
   STX  : constant CHARACTER := stx;    ETX  : constant CHARACTER := etx;
   EOT  : constant CHARACTER := eot;    ENQ  : constant CHARACTER := enq;
   ACK  : constant CHARACTER := ack;    BEL  : constant CHARACTER := bel;
   BS   : constant CHARACTER := bs;     HT   : constant CHARACTER := ht;
   LF   : constant CHARACTER := lf;     VT   : constant CHARACTER := vt;
   FF   : constant CHARACTER := ff;     CR   : constant CHARACTER := cr;
   SO   : constant CHARACTER := so;     SI   : constant CHARACTER := si;
   DLE  : constant CHARACTER := dle;    DC1  : constant CHARACTER := dc1;
   DC2  : constant CHARACTER := dc2;    DC3  : constant CHARACTER := dc3;
   DC4  : constant CHARACTER := dc4;    NAK  : constant CHARACTER := nak;
   SYN  : constant CHARACTER := syn;    ETB  : constant CHARACTER := etb;
   CAN  : constant CHARACTER := can;    EM   : constant CHARACTER := em;
   SUB  : constant CHARACTER := sub;    ESC  : constant CHARACTER := esc;
   FS   : constant CHARACTER := fs;     GS   : constant CHARACTER := gs;
   RS   : constant CHARACTER := rs;     US   : constant CHARACTER := us;
   DEL  : constant CHARACTER := del;
```

```
    -- Other characters:

    EXCLAM     : constant CHARACTER := '!';     QUOTATION  : constant CHARACTER := '"';
    SHARP      : constant CHARACTER := '#';     DOLLAR     : constant CHARACTER := '$';
    PERCENT    : constant CHARACTER := '%';     AMPERSAND  : constant CHARACTER := '&';
    COLON      : constant CHARACTER := ':';     SEMICOLON  : constant CHARACTER := ';';
    QUERY      : constant CHARACTER := '?';     AT_SIGN    : constant CHARACTER := '@';
    L_BRACKET  : constant CHARACTER := '[';     BACK_SLASH : constant CHARACTER := '\';
    R_BRACKET  : constant CHARACTER := ']';     CIRCUMFLEX : constant CHARACTER := '^';
    UNDERLINE  : constant CHARACTER := '_';     GRAVE      : constant CHARACTER := '`';
    L_BRACE    : constant CHARACTER := '{';     BAR        : constant CHARACTER := '|';
    R_BRACE    : constant CHARACTER := '}';     TILDE      : constant CHARACTER := '~';

    -- Lower case letters:

    LC_A : constant CHARACTER := 'a';
    ...
    LC_Z : constant CHARACTER := 'z';

end ASCII;


-- Predefined subtypes:

subtype NATURAL  is INTEGER range 0 .. INTEGER'LAST;
subtype POSITIVE  is INTEGER range 1 .. INTEGER'LAST;

-- Predefined string type:

type STRING is array(POSITIVE range <>) of CHARACTER;

pragma PACK(STRING);


-- The predefined operators for this type are as follows:

-- function "=" (LEFT, RIGHT : STRING) return BOOLEAN;
-- function "/=" (LEFT, RIGHT : STRING) return BOOLEAN;
-- function "<" (LEFT, RIGHT : STRING) return BOOLEAN;
-- function "<=" (LEFT, RIGHT : STRING) return BOOLEAN;
-- function ">" (LEFT, RIGHT : STRING) return BOOLEAN;
-- function ">=" (LEFT, RIGHT : STRING) return BOOLEAN;

-- function "&" (LEFT : STRING;     RIGHT : STRING)     return STRING;
-- function "&" (LEFT : CHARACTER;  RIGHT : STRING)     return STRING;
-- function "&" (LEFT : STRING;     RIGHT : CHARACTER)  return STRING;
-- function "&" (LEFT : CHARACTER;  RIGHT : CHARACTER)  return STRING;

type DURATION is delta  2#1.0#E-14  range -131072..131071.99993896484375;

-- The predefined operators for the type DURATION are the same as for any fixed point type.
```

```
   -- The predefined exceptions:

   CONSTRAINT_ERROR : exception;
   NUMERIC_ERROR    : exception;
   PROGRAM_ERROR    : exception;
   STORAGE_ERROR    : exception;
   TASKING_ERROR    : exception;

end STANDARD;
```

Certain aspects of the predefined entities cannot be completely described in the language itself. For example, although the enumeration type BOOLEAN can be written showing the two enumeration literals FALSE and TRUE, the short-circuit control forms cannot be expressed in the language.

Note:

The language definition predefines the following library units:

- The package CALENDAR (see 9.6)

- The package SYSTEM (see 13.7)
- The package MACHINE_CODE (if provided) (see 13.8)
- The generic procedure UNCHECKED_DEALLOCATION (see 13.10.1)
- The generic function UNCHECKED_CONVERSION (see 13.10.2)

- The generic package SEQUENTIAL_IO (see 14.2.3)
- The generic package DIRECT_IO (see 14.2.5)
- The package TEXT_IO (see 14.3.10)
- The package IO_EXCEPTIONS (see 14.5)
- The package LOW_LEVEL_IO (see 14.6)

APPENDIX C

TEST PARAMETERS

Certain tests in the ACVC make use of implementation-dependent values, such as the maximum length of an input line and invalid file names. A test that makes use of such values is identified by the extension .TST in its file name. Actual values to be substituted are identified by names that begin with a dollar sign. A value is substituted for each of these names before the test is run. The values used for this validation are given below.

Name and Meaning	Value
$BIG_ID1 Identifier of size MAX_IN_LEN with varying last character.	(1..79 => 'A', 80 => '1')
$BIG_ID2 Identifier of size MAX_IN_LEN with varying last character.	(1..79 => 'A', 80 => '2')
$BIG_ID3 Identifier of size MAX_IN_LEN with varying middle character.	(1..32 | 34..80 => 'A', 33 => '3')
$BIG_ID4 Identifier of size MAX_IN_LEN with varying middle character.	(1..32 | 34..80 => 'A', 33 => '4')
$BIG_INT_LIT An integer literal of value 298 with enough leading zeroes so that it is MAX_IN_LEN characters long.	(1..77 => '0', 78..80 => "298")

TEST PARAMETERS

Name and Meaning	Value
$BIG_REAL_LIT A real literal that can be either of floating- or fixed-point type, has value 690.0, and has enough leading zeroes to be MAX_IN_LEN characters long.	(1..74 => '0', 75..80 => "69.0E1")
$BLANKS Blanks of length MAX_IN_LEN - 20	(1..60 => ' ')
$COUNT_LAST Value of COUNT'LAST in TEXT_IO package.	2_147_483_647
$EXTENDED_ASCII_CHARS A string literal containing all the ASCII characters with printable graphics that are not in the basic 55 Ada character set.	"abcdefghijklmnopqrstuvwxyz" & "!$%?@[\]^`{}~"
$FIELD_LAST Value of FIELD'LAST in TEXT_IO package.	255
$FILE_NAME_WITH_BAD_CHARS An illegal external file name that either contains invalid characters or is too long.	F_#$.BAD
$FILE_NAME_WITH_WILD_CARD_CHAR An external file name that either contains a wild card character or is too long.	FILENAME2.BAD
$GREATER_THAN_DURATION A universal real value that lies between DURATION'BASE'LAST and DURATION'LAST or any value in the range of DURATION.	100_000.0
$GREATER_THAN_DURATION_BASE_LAST The universal real value that is greater than DURATION'BASE'LAST.	200_000.0
$ILLEGAL_EXTERNAL_FILE_NAME1 Illegal external file name.	F_#$.BAD
$ILLEGAL_EXTERNAL_FILE_NAME2 Illegal external file names.	FILENAME2.BAD

Name and Meaning	Value
$INTEGER_FIRST The universal integer literal expression whose value is INTEGER'FIRST.	-2_147_483_648
$INTEGER_LAST The universal integer literal expression whose value is INTEGER'LAST.	2_147_483_647
$LESS_THAN_DURATION A universal real value that lies between DURATION'BASE'FIRST and DURATION'FIRST or any value in the range of DURATION.	-100_000.0
$LESS_THAN_DURATION_BASE_FIRST The universal real value that is less than DURATION'BASE'FIRST.	-200_000.0
$MAX_DIGITS Maximum digits supported for floating-point types.	15
$MAX_IN_LEN Maximum input line length permitted by the implementation.	80
$NAME A name of a predefined numeric type other than FLOAT, INTEGER, SHORT_FLOAT, SHORT_INTEGER, LONG_FLOAT, or LONG_INTEGER.	TINY_INTEGER
$NEG_BASED_INT A based integer literal whose highest order nonzero bit falls in the sign bit position of the representation for SYSTEM.MAX_INT.	16#FFFF_FFFE#
$NON_ASCII_CHAR_TYPE An enumerated type definition for a character type whose literals are the identifier NON_NULL and all non-ASCII characters with printable graphics.	(NON_NULL)

APPENDIX D

WITHDRAWN TESTS

Some tests are withdrawn from the ACVC because they do not conform to the Ada Standard. When testing was performed, the following 16 tests had been withdrawn at the time of validation testing for the reasons indicated:

- B4A010C: The object_declaration in line 18 follows a subprogram body of the same declarative part.

- B83A06B: The Ada Standard 8.3(17) and AI-00330 permit the label LAB_ENUMERAL of line 80 to be considered a homograph of the enumeration literal in line 25.

- BA2001E: The Ada Standard 10.2(5) states: "Simple names of all subunits that have the same ancestor library unit must be distinct identifiers." This test checks for the above condition when stubs are declared. However, the Ada Standard does not preclude the check being made when the subunit is compiled.

- BC3204C: The file BC3204C4 should contain the body for BC3204C0 as indicated in line 25 of BC3204C3M.

- C35904A: The elaboration of subtype declarations SFX3 and SFX4 may raise NUMERIC_ERROR (instead of CONSTRAINT_ERROR).

- C41404A: The values of 'LAST and 'LENGTH are incorrect in IF statements from line 74 to the end of the test.

- C48008A: This test requires that the evaluation of default initial values not occur when an exception is raised by an allocator. However, the Language Maintenance Committee (LMC) has ruled that such a requirement is incorrect (AI-00397/01).

WITHDRAWN TESTS

- C4A014A: The number declarations in lines 19-22 are incorrect because conversions are not static.

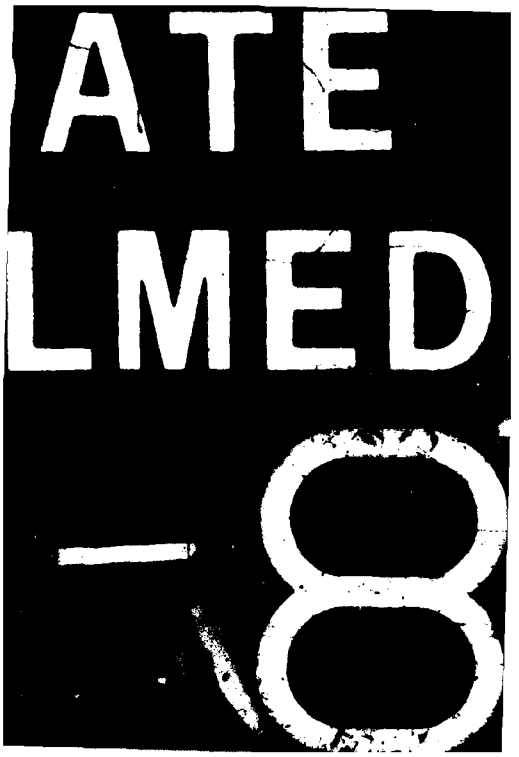
- C92005A: At line 40, "/=" for type PACK.BIG_INT is not visible without a USE clause for package PACK.

- C940ACA: This test assumes that allocated task TT1 will run prior to the main program, and thus assign SPYNUMB the value checked for by the main program; however, such an execution order is not required by the Ada Standard, so the test is erroneous.

- CA1003B: This test requires all of the legal compilation units of a file containing some illegal units to be compiled and executed. According to AI-00255, such a file may be rejected as a whole.

- CA3005A..D (4 tests): No valid elaboration order exists for these tests.

- CE2107E: This test has a variable, TEMP_HAS_NAME, that needs to be given an initial value of TRUE.


ATE
LMED
-8